基于零拷贝的脉冲星 GPU 相干消色散算法*

王博群[1,2]，张海龙[1,2,3,4*]，王杰[1,2,4]，冶鑫晨[1,4]，王万琼[1]，李嘉[1]，张萌[1,2]，张亚州[1,2]，付鹏飞[1,2]

(1. 中国科学院新疆天文台，新疆 乌鲁木齐，830011；2. 中国科学院大学，北京，100049；3. 中国科学院射电天文重点实验室，江苏 南京 210008；4.国家天文科学数据中心，北京，100101)

摘 要：射电脉冲信号在传输过程中受到星际介质的影响会导致轮廓展宽和变形，在研究过程中需要对信号进行消色散处理。本文设计并实现了基于零拷贝的脉冲星数据 GPU 相干消色散算法，采用设备内存映射以消除主机到设备的拷贝开销，利用 CUDA 的 cuFFT 库进行多 BATCH 傅里叶变换以提高 DFT 效率，同时采用多线程实现了传递函数的加速计算。实验结果表明，与传统 CPU 及 GPU 算法相比，本文提出的算法在大数据量时表现良好。

关键词：零拷贝；相干消色散；GPU；CUDA；cuFFT

中图分类号：P161.4　　**文献标识码**：A　　**文章编号**：

0 引言

脉冲星是快速旋转的中子星，具有极强磁场，于 1967 年被 Bell 女士发现。自此，射电脉冲星成为现代天文学的重要研究方向之一。以射电脉冲星作为工具，可以开展高精度计时和授时、天体动力学和天体测量、强场下的引力物理、太阳系外行星、星系和星际介质、超致密物质以及极端环境下的等离子物理等方面的研究[1]。近年来，利用毫秒脉冲星进行引力波探测、脉冲星导航等项目逐渐兴起，高精度脉冲星的观测技术研究成为射电天文学的热门问题。

脉冲星的辐射极其微弱，要求射电望远镜具备很高的灵敏度。建造大型天线、降低接收机的系统噪声和增加接收机的频带宽度是提高灵敏度的重要方法。其中增大观测带宽是经济、可行的策略，但是带宽不能无限增大。增大观测带宽，一方面会给后续处理带来巨大压力，另一方面星际介质会对脉冲星信号产生严重的色散影响，造成脉冲轮廓的展宽和变形，严重时甚至可能导致无法观测到脉冲轮廓，因此在进行脉冲星相关研究时，必须对脉冲星信号进行消色散处理。

近年来，基于图形处理器(Graphic Processing Unit, GPU)的并行计算技术已成为高性能计算领域的研究热点，利用图形处理器可大大提升科学分析、仿真等方面应用程序的运行速度[2]。NVIDIA 推出的统一计算设备架构(Compute Unified Device Architecture, CUDA)可使 GPU 解决相对复杂的问题[3]，为计算平台的构建提供了保障。国际上，使用 CPU+GPU 的

* 基金项目：国家自然科学基金(11873082,11803080)；国家重点研发计划(2018YFA0404704)；中国科学院青年创新促进会；国家天文科学数据中心，中科院科学数据中心体系资助.

收稿日期：2020-00-00；修订日期：2020-00-00

作者简介：王博群，男，硕士研究生，研究方向：数据密集型计算。 Email : wangboqun@xao.ac.cn

通讯作者：张海龙，男，正高级工程师，研究方向：数据密集型研究。 Email:zhanghailong@xao.ac.cn

异构计算平台进行脉冲信号的实时处理已成为主流。通过使用 GPU 对观测数据进行消色散处理，可降低 CPU 负载，并显著提高计算系统性能。

本文针对脉冲星观测数据，首先分析了 GPU 消色散工作原理，然后通过实验研究了 GPU 消色散算法的性能瓶颈，最后设计并实现了基于零拷贝的 GPU 相干消色散算法，并在实验中验证了算法性能。

1 色散与消色散算法

星际介质是低温的等离子体，脉冲星的电磁辐射通过星际介质到达地球时会产生色散，使得高频成分先到达，低频成分后到达。色散是脉冲信号的最重要特征，常用色散度(Dispersion Measure，DM)来表示，它是脉冲星与地球的视向距离上的电子密度的积分，可由公式(1)表示。

$$DM = \int_0^d n_e dl \tag{1}$$

其中，n_e是电子数密度，d为脉冲星与观测望远镜的视向距离。

由星际介质导致频率f_1, f_2的观测时延Δt如公式(2)所示。

$$\Delta t = k_{DM} \cdot DM \cdot (f_1^{-2} - f_2^{-2}) \tag{2}$$

脉冲星信号消色的方法有两种，即非相干消色散和相干消色散。非相干消色散把观测带宽为 BW 的时域信号由滤波器分成若干狭窄通道，然后在时域上把每个通道信号按公式(2)提供的时延量进行平移，最后将所有通道信号幅度累加得到最终的信号序列错误!未找到引用源。。相干消色散将星际介质对脉冲星信号的影响等效为滤波器，通常的做法是将望远镜接收的信号进行傅里叶变换，然后乘以等效滤波器传递函数的反函数即 chirp 函数，再将结果进行傅里叶逆变换，即可得到时域上的脉冲星原始信号[5]。

星际介质传递函数可分解为幅度响应和频率响应。等效滤波器需要同时对信号的幅度和相位进行修正，由此可得离散 chirp 函数的表达式[6](3)。

$$chirp = T_k H_k^{-1} = \frac{1}{N}\left[\left(\frac{f_k}{0.47B}\right)^{80}\right]^{-\frac{1}{2}} \cdot \exp\left[-i\left(\frac{\pm DM f_k^2}{(2.41\times10^{-10}) f_c^2 (f_c \pm f_k)}\right)\right] \tag{3}$$

其中：

$$f_k = \begin{cases} \left(\frac{k}{N}\right)B & for \quad 0 \le k < \frac{N}{2} \\ \frac{(k-N)}{N}B & for \quad \frac{N}{2} \le k < N \end{cases}$$

T_k为 taper 函数，其功能是对脉冲信号的幅度进行修正。H_k^{-1}是脉冲信号传递函数相位响应的反函数，其功能是对相位进行修正。其中 N 为离散信号的频域点数，f_c是观测的中心频率，f_k是频谱中第k点的频率，B为观测带宽。

2 零拷贝

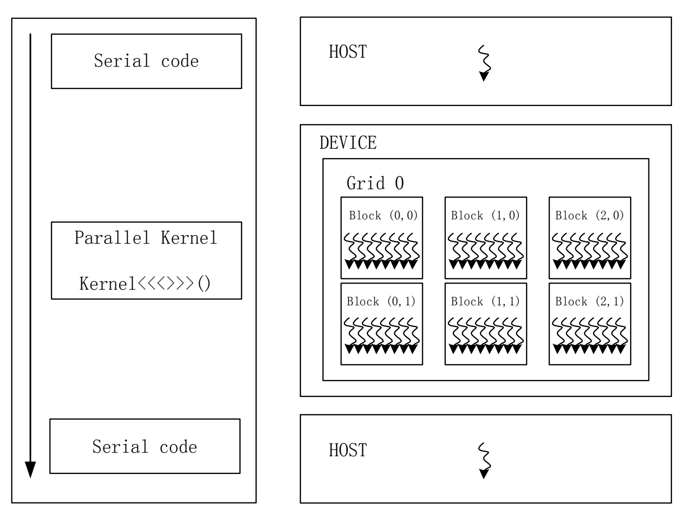


图 1 异构程序执行模型

Fig.1 Heterogeneous program execution model

CPU+GPU 的异构程序执行模型如图 1 所示，其核心思想是将串行代码放置在主机端(CPU)上执行，而将并行代码放置在设备端(GPU)上执行，借助 GPU 多线程来减少计算开销。该模型假定主机和设备都在 DRAM 中保持自己独立的储存空间，分别称为主机存储器和设备存储器。在程序执行时，需要将预处理数据从主机存储器拷贝到设备存储器，而在 GPU 处理结束后，还要将运行结果拷贝回主机存储器，这造成不小的通信开销。

```
float *H,*H_MAP;
...
cudaDeviceProp prop;
cudaGetDeviceProperties(&prop,0);
if(!prop.canMapHostMemory){
        exit(0);
}
cudaSetDeviceFlags(cudaDeviceMapHost);
cudaHostAlloc(&H,nBytes,cudaHostAllocMapped);
cudaHostGetDevicePointer(&H_MAP,H,0);
Kernel<<<gridSize,blockSize>>>(H_MAP);
```

图 2 零拷贝流程

Fig.2 The process of Zero-copy

零拷贝是指 GPU 直接映射主机内存并通过 PCIe 访问而无需进行显式内存传输[7]。它需要使用锁页映射内存，核函数线程可直接访问主机内存，从而减少内存拷贝开销。但因为数据不会在 GPU 中进行缓存，所以锁页映射内存只能被读取和写入一次，多次读写会降低其效率，这是零拷贝的局限性。

零拷贝的代码如图 2 所示，其中 cudaGetDeviceProperties()返回的 prop 通过调用 canMapHostMemory 来检查当前 GPU 是否支持将主机内存映射到设备的地址空间。通过使用 cudaDeviceMapHost 调用 cudaSetDeviceFlags()来启动锁页内存映射。然后使用

cudaHostAlloc()分配锁页映射主机内存，并通过函数 cudaHostGetDevicePointer()获得指向映射设备地址空间的指针。在零拷贝代码中，Kernel()可以使用指针 H_MAP 引用锁页主机内存，其使用方式与调用 H_MAP 操作设备内存完全相同。

3 基于零拷贝的相干消色散算法

望远镜在接收脉冲星信号时会受到其他信号的干扰，这种射频干扰(RFI)可定义为由人类活动造成的不良射电信号。RFI 轻则会降低脉冲星数据的质量，影响后续的科学研究；重则会造成脉冲信号丢失，导致有用的信号完全淹没在噪声中。因此必须要对观测数据进行 RFI 消减。RFI 可以分为窄带干扰与宽带干扰，全球定位系统(GPS)的通信信号、飞机通讯信号及 FM 广播信号等都属于窄带干扰，它们在频域上是窄带的，但因为持续时间长，所以在时域上长期存在;雷电、高压电缆、电子围栏和其他能够引起瞬态电泳的信号属于宽带干扰，它们在时域上是局部的，但会污染多个频域通道并因此表现出宽带特性。

由于宽频带干扰在频域内不易识别，而在时域内表现为短时强脉冲,所以在时域进行宽带 RFI 的消减，具体方法是：

(1) 每次取数据块大小为B，由n个点组成，即 $B_1, B_2 \dots B_n$。

(2) 求数据块值的均方根(rms)。

(3) 使用均方根来替代B中值超过$s \cdot rms$的数据，其中s是设置的全局宽带阈值。

其公式描述如(4)所示：

$$B_k = \sqrt{\frac{\sum_{i=1}^{n} B_i^2}{n}}, \quad B_k \geq s \cdot rms \tag{4}$$

窄带干扰在时间上是连续存在的，在时域上很难将其分离，而它在频域上表现为某些突出的尖峰，所以在频域进行窄带 RFI 的消减，具体方法是：

(1) 将经过宽带 RFI 消减的数据块B变换至频域并用F表示，大小为m，$m = n/2 + 1$。

(2) 计算F中的每一个点$F_1, F_2 \dots F_m$的模值$|F_1|, |F_2| \dots |F_m|$，然后求模值的平均值($mean$)。

(3) 生成长度为m的滤波器K，如果$|F_i|$大于$u \cdot mean$，令$K_i = 0$，否则$K_i = 1$，其中u是设置的全局窄带阈值。

(4) 将K与F相乘，得到消除窄带 RFI 后的频域序列。

其公式描述如(5)所示：

$$F = F \cdot K \tag{5}$$

其中

$$K_i = \begin{cases} 0 & |F_i| > u \cdot mean, 1 \leq i \leq m \\ 1 & else \end{cases}$$

$$mean = \frac{\sum_{i=1}^{m} |F_i|}{m}$$

在相干消色散算法中，需要将时域序列变换至频域进行处理，这就要求对数据进行离散傅里叶变换(DFT)。为了使傅里叶变换更加高效，首先需要对原始数据进行分块处理，然后分别将每一块进行 DFT，乘以星际介质的传递函数进行消色散后，再进行逆变换(IDFT)，得到最终的时间序列。

传统的 CPU 使用 FFTW 进行 DFT，FFTW 是一个快速计算离散傅里叶变换的标准 C 程序集，是目前公认速度最快的开源 FFT 变换程序[9]。在 NVIDIA GPU 上进行离散傅里叶变换需要使用 cuFFT[9]，cuFFT 提供了经过大量优化和广泛测试的 FFT 程序库，使得用户可以借助 GPU

强大的计算能力进行快速变换。

```
# define BATCH 10
# define N 256
...
        cufftComplex *host_in,*device_in;
        cufftReal *host_out,*device_out;
        cufftHandle plan;
        cufftPlan1d(&plan,N,CUFFT_R2C,BATCH);
        ...
        cufftExecR2C(plan,device_in,device_out);
        ...
```

图 4 R2C DFT 核心代码

Fig.4 The core code of R2C DFT

图 4 给出从实数到复数的 GPU DFT 变换核心代码，首先定义处理的批次 BATCH 和每一批的数据量 N。然后分别定义 cufftComplex 和 cufftReal 类型的指针，host_in 和 host_out 分别代表主机端(CPU)输入输出数组，而 device_in 和 device_out 代表了设备端(GPU)的输入输出数组。使用 cufftHandle 定义 cuFFT 句柄，即一个执行计划。接着使用一维的 cufftPlan1d 对 plan 进行构造，通过使用 BATCH 和 N 进行批次控制来告知 GPU 需要进行多少次一维变换和每次变换的数据量，实现了分别对不同块进行 DFT。

之后为主机端数组申请锁页映射内存，使用零拷贝完成内存到显存的映射，节省了内存拷贝开销，其核心代码见图 2。内存分配完毕后，使用 cufftExecR2C 执行傅里叶变换并将结果写入 device_out 中。值得注意的是，cuFFT 在执行 R2C 时会自动去除 DFT 的对称部分，即如果实数的个数为 N，那么得到的复数只有 N/2+1 个。

为了最大化指令吞吐量，核函数中的数学计算全部使用 CUDA 提供的固有函数。使用 sinf 和 cosf 来替换传统的 sin 和 cos，前者能够在不影响最终结果的前提下加快运行速度。此外，根据 David Střelák 等的建议[11]，在 DFT 运行过程中采取以下的优化策略：

(1) 通过对数据进行截取，保证数据长度是 2 的幂级数，加快了运行速度。

(2) 通过使用单精度变换，使用额外空间进行 DFT 结果缓存(out of place)，提高了运行效率。

(3) 通过使用多批次变换进行 DFT，而不是重复使用 cufftPlan1d，缩短了运行时间。

4 实验结果

算法测试所使用的开发设备如表 1 所示，操作平台为 Windows 10，开发工具为 Visual Studio 2015、NVIDIA Visual Profiler，开发语言为 C、CUDA C、Python。

表 1 开发设备

Tab.1 Development equipment

Name	Device Type	Storage	Frequency
CPUA	Intel Core i7-4720HQ	8GB	2.6GHZ
GPUA	NVIDIA GeForce GTX 960M	4GB	1.1GHZ
CPUB	Intel Core i9-9900K	64GB	3.6GHZ
GPUB	NVIDIA GeForce RTX 2080 SUPER	8GB	1.65GHZ

使用真实的脉冲星基带数据进行算法测试。数据格式为双极化 8 比特采样的 PSRDADA，

观测源为 J0437-4715，观测信息如表 2 所示。基带数据记录时间约为 8s，大小为 12.8GB。

表 2 J0437-4715 观测信息

Tab. 2 J0437-4715 Observation information

J0437-4715	Information
Period	0.005757451936712637s
DM	2.64476
Center frequency	1382MHZ
Observation bandwidth	400MHZ
Telescope	Parkes

为验证本文算法的正确性，进行以下实验：首先对 11.72GB 的原始数据进行折叠，生成轮廓图作为对照。然后固定数据块大小为 16MB，使用本文算法分别处理 50、250、750 个数据块，其数据量分别为 0.78GB、3.91GB、11.72GB，最后生成不同情况下的轮廓图。实验结果如图 5 所示，各子图横轴表示脉冲相位，纵轴表示归一化振幅。(a)代表原始数据折叠未进行消色散，(b)、(c)、(d)分别代表数据块个数为 50、数据块个数为 250、数据块个数为 750 时消色散后得到的脉冲轮廓。

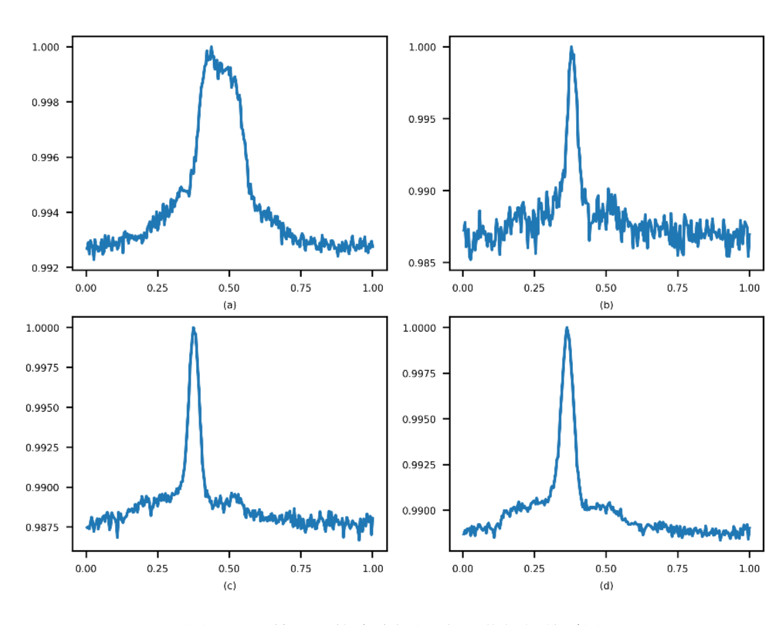


图 5 不同情况下的脉冲相位-归一化振幅关系图

Fig. 5 Pulse phase -Normalized amplitude relationship in different situations

对比图 5 中的(a)、(d)可得，本文算法可以正确的对脉冲信号进行修正，完整的展现出脉冲轮廓。通过与新疆天文台数据中心的 J0437-4715 轮廓进行比对，证明所得轮廓的正确性。对比(b)、(c)、(d)可知，随着数据块数量的不断增加，所得脉冲轮廓信噪比逐渐提高。

为验证 CPU 算法、传统 GPU 算法以及本文算法(ZGPU)在不同数据量下的执行速度，进行以下实验：首先固定数据块大小为 8M，将每次处理的数据块数量分别设置为 10、50、100、200、400、800、1500，对应数据量为 0.08GB、0.39GB、0.78GB、1.56GB、3.13GB、6.25GB、11.72GB，然后令三种方法分别在每个数据量下运行十次，最后取平均值作为运行时间。此

外，为验证设备性能对算法的影响，本实验采用两套设备进行测试，设备型号如表 1 所示。实验结果如表 3 所示，时间单位为 s。

表 3 实验结果

Tab.3 Experimental results

Equipment	Model	Algorithm	10	50	100	200	400	800	1500
CPU	CPUA	CPU	9.213	42.351	84.024	165.608	330.420	659.481	1259.977
	CPUB	GPU	5.864	26.270	51.756	102.566	205.499	408.408	768.446
GPU	GPUA	GPU	2.142	6.351	11.799	23.301	47.097	94.287	180.296
		ZGPU	2.054	6.053	11.230	21.734	43.100	87.178	166.208
	GPUB	GPU	0.830	2.248	4.018	7.561	14.635	28.794	53.603
		ZGPU	0.808	2.130	3.781	7.077	13.680	26.859	49.945

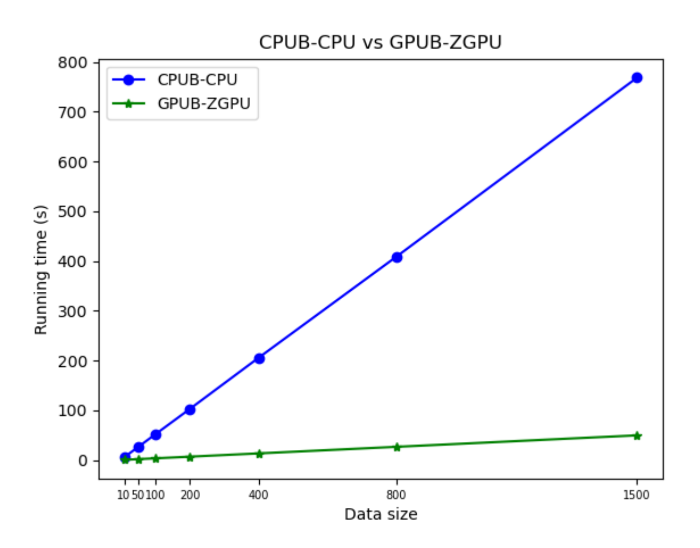


图 6 CPUB-CPU 与 GPUB-ZGPU 数据量-运行时间关系图

Fig.6 CPUB-CPU vs GPUB-ZGPU Data size-Running time relationship

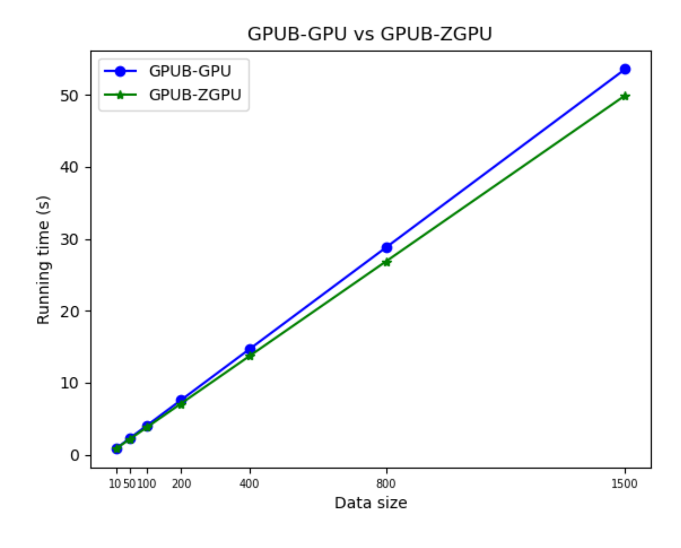


图 7 GPUB-GPU 与 GPUB-ZGPU 数据量-运行时间关系图

Fig.7 GPUB-GPU vs GPUB-ZGPU Data size-Running time relationship

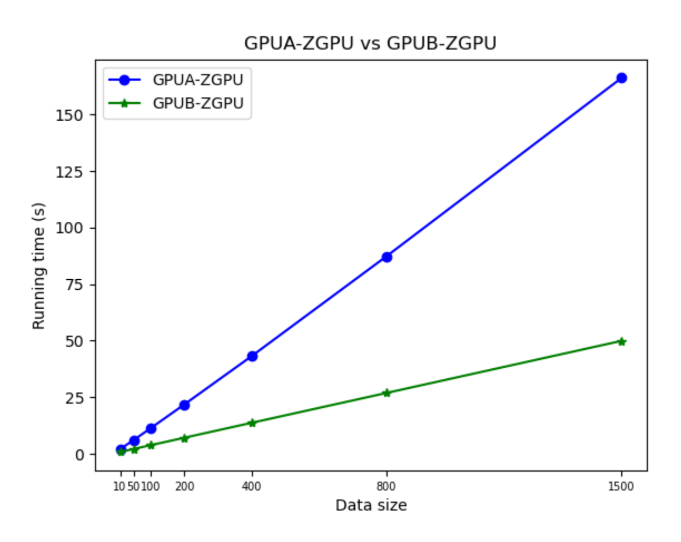


图 8 GPUA-ZGPU 与 GPUB-ZGPU 数据量-运行时间关系图

Fig.8 GPUA-ZGPU vs GPUB-ZGPU Data size-Running time relationship

本文采用 Model-Algorithm 的方式对算法进行讨论，例如 GPUA-GPU 即代表运行在 GPUA 上的传统 GPU 算法。从表 3 中抽取 CPUB-CPU 与 GPUB-ZGPU 做折线图，得到图 6;取 GPUB-GPU 与 GPUB-ZGPU 做折线图，得到图 7;取 GPUA-ZGPU 和 GPUB-ZGPU 得到图 8。

结合表 3 分析图 6 可知：两种算法的执行时间都随着数据量增加而线性增长，但是传统 CPU 算法的平均运行时间远大于本文算法。在数据大小为 11.72GB 时，传统 CPU 算法需要 768.446s 才能完成消色散，而本文算法仅需 49.945s，速度提升了约 15 倍。由此可得，本文算法在速度上优于传统 CPU 算法。

结合表 3 分析图 7 可知：两种算法的执行时间都随着数据量增加而线性增长，在数据量较小时，本文算法与传统 GPU 算法的差距不明显，但随着数据量的增大，两种算法的时间差也在线性增长。当数据量分别为 400、800、1500 时，两种算法的时间差为 0.949s、1.935s、3.658s，即数据量增加 2 倍，时间差也增加 2 倍。由此可得，本文算法在数据量较大时优于传统 GPU 算法，数据量越大，本文算法的优势就越明显。

结合表 3 分析图 8 可知：本文算法在新架构 GPU 上的平均运行时间明显比之前的 GPU 短。在数据量为 1500 时，本文算法在 GPUA 上需要运行 166.208s，而在 GPUB 上仅需 49.945s，速度提高了约 3 倍。由此可得，硬件性能是影响本文算法的重要因素。

5 结论

本文设计实现了基于零拷贝的 GPU 脉冲星相干消色散算法。通过合理构造核函数实现了最大化并行执行，采取零拷贝技术提高了内存吞吐量，使用 CUDA 内置计算函数优化了指令调用。通过在 DFT 中对 cuFFT 进行优化，提高了其运行效率。结合真实数据测试表明本文实现的算法优于传统的 CPU 算法，且在数据量较大时优于传统的 GPU 算法。基于零拷贝的 GPU 消色散算法为高效实现脉冲星数据处理相关研究提供了一种新的解决思路。实验数据及源代码可在新疆天文台数据中心[1]获取。

[1] http://data.xao.ac.cn/static/zerocopy.rar

致谢：感谢国家自然科学基金(11873082,11803080)；国家重点研发计划(2018YFA0404704)；中国科学院青年创新促进会；国家天文科学数据中心，中科院科学数据中心体系的资助。本论文得到中国虚拟天文台、国家天文科学数据中心、中科院科学数据中心体系提供的数据资源和技术支持。

参考文献：

[1] 徐永华，罗近涛，李志玄，等. 基于 MARK5B+ DSPSR 的基带数字脉冲星观测系统[J]. 天文研究与技术, (Xu Yonghua, Luo Jintao, Li Zhixuan, Hao Longfei, Wang Min, Dong Jiang. A Pulsar Observation System with a Mark 5B Recording System and a DSPSR Software)2013, 10(4): 352-358.

[2] Gu J, Chowdhury M, Shin K G, et al. Tiresias: A {GPU} cluster manager for distributed deep learning[C]//16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19). 2019: 485-500.

[3] Nvidia C. CUDA C programming guide, version 10.1[J]. NVIDIA Corp, 2019.

[4] 黄玉祥，汪敏，郝龙飞，李志玄，徐永华. 脉冲星信号相干消色散与非相干消色散的比较研究[J]. 天文研究与技术,(Huang Yuxiang, Wang Min, Hao Longfei, Li Zhixuan, Xu Yonghua. Comparative Study between the Coherent De-dispersion and the Incoherent De-dispersion of Pulsar Signal. Astronomical Research & Technolog) 2019, 16(1): 16-24.

[5] 王显海. 宽带实时脉冲星数字接收机关键技术研究[D]. 东南大学,(Wang Xianhai. Research on the key techniques for the wideband real-time pulsar digital receiver. Southeast University) 2016.

[6] Stairs I H. Observations of Binary and Millisecond Pulsars with a Baseband Recording System[D]. Princeton University, 1998.

[7] Profiler N. NVIDIA visual profiler–NVIDIA developer zone[J]. 2014.

[8] Bracewell R N, Bracewell R N. The Fourier transform and its applications[M]. New York: McGraw-Hill, 1986.

[9] Frigo M, Johnson S G. FFTW: An adaptive software architecture for the FFT[C]//Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181). IEEE, 1998, 3: 1381-1384.

[10] NVIDIA. CUDA Fast Fourier Transform library (cuFFT)[J]. 2018.

[11] Střelák D, Filipovič J. Performance analysis and autotuning setup of the cuFFT library[C]//Proceedings of the 2nd Workshop on AutotuniNg and aDaptivity AppRoaches for Energy efficient HPC Systems. 2018: 1-6.

Coherent de-dispersion algorithm for pulsar GPU based on zero-copy

WANG BoQun[1,2], ZHANG HaiLong[1,2,3,4*], WANG Jie[1,2,4], YE XinChen[1,4], WANG WanQiong[1], LI Jia[1], ZHANG Meng[1,2], ZHANG YaZhou[1,2], FU PengFei[1,2]

(1. Xinjiang Astronomical Observatory, Chinese Academy of Sciences, Urumqi 830011, China；

2. University of Chinese Academy of Sciences, Beijing 100049, China；

3. Key Laboratory of Radio Astronomy, Chinese Academy of Sciences, Nanjing 210008, China；

4. National Astronomical Data Center, Beijing 100101, China)

Abstract: The radio pulse signal is affected by the interstellar medium during the transmission process, which will cause the profile to be broadened and deformed. In the research process, the signal needs to be de-dispersed. This paper designed and implemented a zero-copy-based GPU coherent de-dispersion algorithm for pulsar data, used device memory mapping to eliminate the copy cost from host to device, and used CUDA's cuFFT library to perform multi-BATCH Fourier transform to improve DFT efficiency. Multithreading was used to realize the accelerated calculation of the transfer function. The experimental results show that, compared with traditional CPU and GPU algorithms, the algorithm proposed in this paper performs well in large amounts of data.

Key words: zero-copy, coherent dedispersion, GPU, CUDA, cuFFT